EXHIBIT A

real-time system, some of the tasks are real-time tasks, and these have a certain degree of urgency to them. Such tasks are attempting to control or react to events that take place in the outside world. Because these events occur in "real time," a real-time task must be able to keep up with the events with which it is concerned. Thus, it is usually possible to associate a deadline with a particular task, where the deadline specifies either a start time or a completion time. Such a task may be classified as hard or soft. A **hard real-time task** is one that must meet its deadline; otherwise it will cause undesirable damage or a fatal error to the system. A **soft real-time task** has an associated deadline that is desirable but not mandatory; it still makes sense to schedule and complete the task even if it has passed its deadline.

Another characteristic of real-time tasks is whether they are periodic or aperiodic. An **aperiodic task** has a deadline by which it must finish or start, or it may have a constraint on both start and finish time. In the case of a **periodic task**, the requirement may be stated as "once per period T" or "exactly T units apart."

Characteristics of Real-time Operating Systems

Real-time operating systems can be characterized as having unique requirements in five general areas [MORG92]:

- Determinism
- Responsiveness
- User control
- Reliability
- Fail-soft operation.

An operating system is **deterministic** to the extent that it performs operations at fixed, predetermined times or within predetermined time intervals. When multiple processes are competing for resources and processor time, no system will be fully deterministic. In a real-time operating system, process requests for service are dictated by external events and timings. The extent to which an operating system can deterministically satisfy requests depends first on the speed with which it can respond to interrupts and, second, on whether the system has sufficient capacity to handle all requests within the required time.

One useful measure of the ability of an operating system to function deterministically is the maximum delay from the arrival of a high-priority device interrupt to when servicing begins. In non-real-time operating systems, this delay may be in the range of tens to hundreds of milliseconds, while in real-time operating systems that delay may have an upper bound of anywhere from a few microseconds to a millisecond.

A related but distinct characteristic is **responsiveness.** Determinism is concerned with how long an operating system delays before acknowledging an interrupt. Responsiveness is concerned with how long, after acknowledgment, it takes an operating system to service the interrupt. Aspects of responsiveness include the following:

1. The amount of time required to initially handle the interrupt and begin execution of the interrupt service routine (ISR). If execution of the requires a

process s
within th

2. The amo
on the ha
3. The effec
another i

Determinism a
events. Respon
systems must r
flows external
User con
in ordinary op
either has no
only provide b
class. In a real
control over ta
soft tasks and t
also allow the
swapping, wha
fer algorithms
have, and so o
Reliabili
time systems.
rebooting the
may result in
replaced. But
Loss or degra
from financial
As in oth
ating system is
to various fail
ity of a system
possible. For e
tion of data wi
the memory c
the system. In
lem or minimi
user or user p
operation perl
sary, an attem
An impc
time system is
system will me
less critical ta
To meet
ically include

these have a certain
:ol or react to events
:cur in "real time," a
/hich it is concerned.
cular task, where the
:h a task may be clas-
ieet its deadline; oth-
› the system. A **soft**
not mandatory; it still
passed its deadline.
y are periodic or ape-
ish or start, or it may
of a **periodic task**, the
y T units apart."

mique requirements in

t it performs operations
: intervals. When multi-
:, no system will be fully
iests for service are dic-
in operating system can
›eed with which it can
ias sufficient capacity to

/stem to function deter-
gh-priority device inter-
/stems, this delay may be
real-time operating sys-
m a few microseconds to

ess. Determinism is con-
:nowledging an interrupt.
:dgment, it takes an oper-
iess include the following

interrupt and begin exe-
ecution of the requires

process switch, then the delay will be longer than if the ISR can be executed within the context of the current process.

2. The amount of time required to perform the ISR. This generally is dependent on the hardware platform.
3. The effect of interrupt nesting. If an ISR can be interrupted by the arrival of another interrupt, then the service will be delayed.

Determinism and responsiveness together make up the response time to external events. Response-time requirements are critical for real-time systems, because such systems must meet timing requirements imposed by individuals, devices, and data flows external to the system.

User control is generally much broader in a real-time operating system than in ordinary operating systems. In a typical non-real-time operating system, the user either has no control over the scheduling function of the operating system or can only provide broad guidance, such as grouping users into more than one priority class. In a real-time system, however, it is essential to allow the user fine-grained control over task priority. The user should be able to distinguish between hard and soft tasks and to specify relative priorities within each class. A real-time system may also allow the user to specify such characteristics as the use of paging or process swapping, what processes must always be resident in main memory, what disk transfer algorithms are to be used, what rights the processes in various priority bands have, and so on.

Reliability is typically far more important for real-time systems than non-real-time systems. A transient failure in a non-real-time system may be solved by simply rebooting the system. A processor failure in a multiprocessor non-real-time system may result in a reduced level of service until the failed processor is repaired or replaced. But a real-time system is responding to and controlling events in real time. Loss or degradation of performance may have catastrophic consequences, ranging from financial loss to major equipment damage and even loss of life.

As in other areas the difference between a real-time and a non-real-time operating system is one of degree. Even a real-time system must be designed to respond to various failure modes. **Fail-soft operation** is a characteristic that refers to the ability of a system to fail in such a way as to preserve as much capability and data as possible. For example, a typical traditional UNIX system, when it detects a corruption of data within the kernel, issues a failure message on the system console, dumps the memory contents to disk for later failure analysis, and terminates execution of the system. In contrast, a real-time system will attempt either to correct the problem or minimize its effects while continuing to run. Typically, the system notifies a user or user process that it should attempt corrective action and then continues operation perhaps at a reduced level of service. In the event a shutdown is necessary, an attempt is made to maintain file and data consistency.

An important aspect of fail-soft operation is referred to as *stability*. A real-time system is stable if, in cases where it is impossible to meet all task deadlines, the system will meet the deadlines of its most critical, highest-priority tasks, even if some less critical task deadlines are not always met.

To meet the foregoing requirements, current real-time operating systems typically include the following features [STAN89]:

- Fast process or thread switch
- Small size (with its associated minimal functionality)
- Ability to respond to external interrupts quickly
- Multitasking with interprocess communication tools such as semaphores, signals, and events
- Use of special sequential files that can accumulate data at a fast rate
- Preemptive scheduling based on priority
- Minimization of intervals during which interrupts are disabled
- Primitives to delay tasks for a fixed amount of time and to pause/resume tasks
- Special alarms and timeouts.

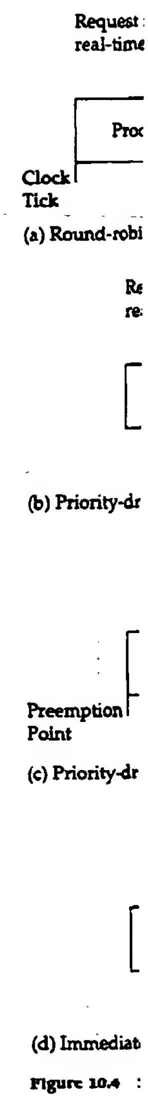
The heart of a real-time system is the short-term task scheduler. In designing such a scheduler, fairness and minimizing average response time are not important. What is important is that all hard real-time tasks complete (or start) by their deadline and that as many as possible soft real-time tasks also complete (or start) by their deadline.

Most contemporary real-time operating systems are unable to deal directly with deadlines. Instead, they are designed to be as responsive as possible to real-time tasks so that, when a deadline approaches, a task can be quickly scheduled. From this point of view, real-time applications typically require deterministic response times in the several-millisecond to submillisecond span under a broad set of conditions; leading-edge applications—in simulators for military aircraft, for example—often have constraints in the range of 10 to 100 μs [ATLA89].

Figure 10.4 illustrates a spectrum of possibilities. In a preemptive scheduler that uses simple round-robin scheduling, a real-time task would be added to the ready queue to await its next time slice, as illustrated in Figure 10.4a. In this case, the scheduling time will generally be unacceptable for real-time applications. Alternatively, in a nonpreemptive scheduler, we could use a priority scheduling mechanism, giving real-time tasks higher priority. In this case, a real-time task that is ready would be scheduled as soon as the current process blocks or runs to completion (Figure 10.4b). This could lead to a delay of several seconds if a slow, low-priority task were executing at a critical time. Again, this approach is not acceptable. A more promising approach is to combine priorities with clock-based interrupts. Preemption points occur at regular intervals. When a preemption point occurs, the currently running task is preempted if a higher-priority task is waiting. This would include the preemption of tasks that are part of the operating system kernel. Such a delay may be on the order of several milliseconds (Figure 10.4c). While this last approach may be adequate for some real-time applications, it will not suffice for more demanding applications. In those cases, the approach that has been taken is sometimes referred to as immediate preemption. In this case, the operating system responds to an interrupt almost immediately, unless the system is in a critical-code lockout section. Scheduling delays for a real-time task can then be reduced to 100 μs or less.

Real-time Scheduling

Real-time scheduling is one of the most active areas of research in computer science. In this subsection, we provide an overview of the various approaches to real-time scheduling and look at two popular classes of scheduling algorithms.



(a) Round-robi

(b) Priority-dr

(c) Priority-dr

(d) Immediat

Figure 10.4